Disk Mirroring with Alternating Deferred Updates

Christos A. Polyzois Anupam Bhide Daniel M. Dias

IBM T. J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
{christos, dias}@watson.ibm.com

Abstract

Mirroring is often used to enhance the reliability of disk systems, but it is usually considered expensive because it duplicates storage cost and increases the cost of writes. Transaction processing applications are often disk arm bound. We show that for such applications mirroring can be used to increase the efficiency of data accesses. The extra disk arms pay for themselves by providing extra bandwidth to the data, so that the cost of the overall system compares favorably with the cost of non-redundant disks. The basic idea is to have the mirrored disks out of phase with one handling reads and the other applying a batch of writes. We also present an efficient recovery procedure that allows reconstruction of a failed disk while guaranteeing the same level of read performance as during normal operation.

1 Introduction

Two goals have driven disk systems research in recent years: reliability and performance. Reliability is typically achieved through redundancy, e.g., mirrored disks or redundant arrays of inexpensive disks (RAID). Mirroring has good reliability characteristics and read performance, but it incurs a disk arm overhead for writes and a significant storage overhead for replicating the data. Compared with RAID, mirroring has simpler and more efficient recovery from disk failure. RAID architectures [15] try to ameliorate the space overhead of mirroring at the expense of increasing disk arm overhead under normal operation and complicating recovery.

The selection among reliable disk architectures depends on the application characteristics. An application may be:

1. **Disk capacity bound:** For example, storage of historical data requires only disk space; neither performance nor reliability is critical. For such applications the goal is minimal $/GB, so that large, cheap disks with inexpensive controllers can be used. Redundancy is not important, since backups can be kept on tapes.

2. **Sequential bandwidth bound:** A logging process is typically sequential bandwidth bound. For such applications file striping improves bandwidth; striping data on a RAID-5 array [15] can provide both high sequential bandwidth and reliability at a low cost.

3. **Random access bound:** On-line transaction processing applications usually make many small random accesses. As indicated in Reference [9], installations often leave as much as 55% of their disk space empty, to ensure that the performance for access to the actual data is satisfactory. Thus, the number of disks required is often determined by the number of arms necessary to support the workload (*disk arm bound*), not by the space occupied by the data. More efficient utilization of the disk arms can increase space utilization and actually reduce the number of disks required.

Our focus in this paper is on case (3) above,

Proceedings of the 19th VLDB Conference
Dublin, Ireland, 1993

i.e., random access bound applications. In Reference [4] it is shown that mirroring is better than RAID-5 in such cases. We show that the performance of mirroring can be enhanced substantially. We exploit the redundancy inherent in mirroring to gain performance by taking advantage of technology trends: large memories and efficient sequential disk I/O.

Disk access times have not improved significantly in the past decade, so research efforts have recently focused on delaying writes and applying them to the disk in batches, in order to benefit from the higher performance of sequential vs. random access. Such efforts include [1, 8, 16, 17, 18, 19]. As large memories become available, it is possible to delay writes and apply them in a batch; therefore, we expect this technique to gain significance. A potential drawback is the response time for read operations: reads must be suspended during the application of a batch, which may in turn affect transaction response time. Our scheme exploits the availability of two mirrored arms, which ensures good read response time by alternating batch updates between the two mirrors. We address batch updates on a single disk in Section 5.

Our goal is to provide both availability and performance. Since redundant data must be stored to ensure availability, this data might as well be used to enhance performance. We start from the observation that mirroring and batch updates are complementary, in the sense that the weak point of batch updates (read response time) is the strong point of mirroring [2, 3]. Careful scheduling of disk accesses in the presence of mirroring can lead to increased disk arm utilization, and thus to lower cost.

Our technique also gives rise to an efficient scheme for recovery from disk failure. In contrast to previous schemes, our method does not hurt read access to data that had a copy on the failed disk.

Section 2 describes our technique during normal operation. Section 3 evaluates the fault-free performance of the proposed technique using a simple analytical model (Section 3.1) and simulation (Section 3.2). Section 4 presents our procedure for recovering from single disk failure and estimates the recovery time. Finally, the region of applicability of the proposed scheme is examined in Section 5.

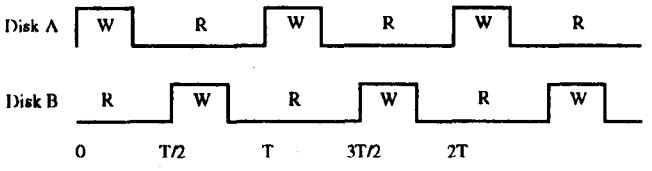

Figure 1: Read/write alternation.

2 Normal Operation

We assume that the system has a large buffer pool. Transaction processing systems typically employ a write-ahead log, so that data pages can be written to their home location on disk asynchronously. For random access bound applications that do not use a write-ahead log, non-volatile, reliable memory may be required. We return to this issue in Section 5. The buffer can be located either in main memory or at the disk controller; our technique is applicable in both cases.

During normal operation, updates are not applied to disks immediately. Periodically, when a certain number of updates have accumulated in the buffer, they are sorted in physical layout order and applied to the disks as a batch. Algorithms for applying updates in batches efficiently have been discussed and analyzed in [17]. The improvement in bandwidth with respect to random accesses can be significant.

While a disk is not applying updates, it services read requests. Thus, each disk alternates between time spans of write-only and read-only activity. Let the period during which the updates accumulate be T, and consider a pair of mirrored disks A and B. Both disks apply the updates in batches. However, they do *not* apply the batches simultaneously. Instead, they operate at a phase difference of 180°. For example, assume that disk A starts applying updates at time 0. Disk B will start applying updates at time $T/2$. Disk A applies the second batch at time T, then disk B applies the second batch at time $3T/2$, and so on. Figure 1 illustrates this alternation with a timing diagram, where the high value represents write activity and the low value represents read activity for the corresponding disk.

If each disk can apply the batch in time less than $T/2$, then the write-only time spans of the two disks do not overlap, so that there is always at least one disk available to service read requests; there may even be some time when both disks are servicing reads. This case is shown in Figure 1. Read requests are routed to the disk operating

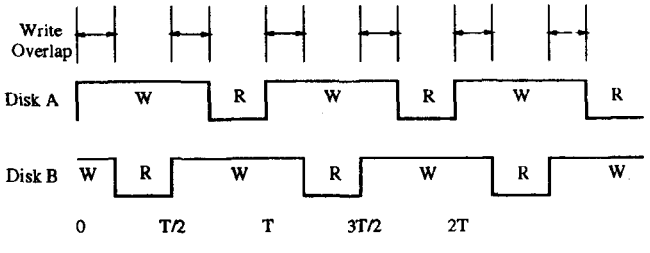

Figure 2: Overlapping write periods.

in read-only mode and are serviced immediately. Thus, batches can be made large to gain efficiency, without affecting read response time. The limiting factor for batch size is the amount of memory available.

In cases of extremely write-intensive loads, it may take each disk longer than $T/2$ to install a batch, so that the write time spans of the two disks will overlap. This situation is shown in Figure 2. Read requests that arrive during the overlap period are deferred until one of the disks finishes its write batch, which hurts read response time. Alternatively, such reads can be serviced immediately, thus disrupting the efficient installation of updates. If the second option is adopted, the reads can be routed to the disk whose head is closest to the cylinder where the read must be performed.

The response time for reads and the throughput for each individual disk in the proposed scheme are at least as good as in the non-redundant case. However, the writes are performed twice, so the total throughput of a mirrored pair is less than twice the throughput of an individual disk. In order to evaluate the efficiency of the overall system, we estimate the number of arms required to support a certain workload and compare this number with the number of arms that would be necessary if no replication was used. We assume a transaction processing type of workload, which is disk arm bound rather than space bound [10]. In such cases, it is common for disks to be used well below their nominal space capacity so that the arms can sustain the I/O bandwidth directed to the stored data. The Fujitsu Eagle disk drive [7] has been used frequently in disk performance studies in the literature [17], so we use the parameters for that disk drive, which are shown in Table 1. In our analysis we focus on non-overlapping write periods; we return to the issue of overlapping write periods in Section 5.

Parameter	Symbol	Value
No of Cylinders	*cyls*	840
No of Surfaces	*surfs*	20
Block Size	*blksiz*	4K
Blocks per Track	$blks_{trk}$	8†
Rotation Time	t_{rot}	16.6 msec
Half Rotation Time	t_{hrot}	8.3 msec
Average Seek	t_{avseek}	18 msec
Block transfer Time	t_{xfer}	2 msec
Avg. Random Access	t_{ran}	28.4 msec
Single Track Seek	$t_{trkseek}$	5.47 msec
End to End Seek	$t_{endseek}$	35 msec
Memory Fraction	f	1%†
Write Ratio	w	0.5†

Table 1: System Parameters (†default values)

3 Evaluation of Fault-Free Performance

3.1 Simple Analytical Model

The analytical model we develop is conservative, i.e., we make pessimistic assumptions to simplify the analysis. The model is actually a hybrid analytic-simulation model, because we use a Monte Carlo simulation as a subroutine to the analytical model.

We assume that the entire system receives I/O requests at a rate λ. A fraction w of these requests are writes. The requests are distributed uniformly across all disks and over all blocks in a given disk. In database systems, transactions tend to be short and usually make requests for small amounts of data (records); records are usually stored in a sequence determined by their contents rather than their frequency of access. Thus, hot (i.e., frequently accessed) data will be scattered across all disks and all blocks. Our assumption may not be true for file systems, which often cluster hot files in the middle of the disk to reduce seek times.

D_{base} denotes the number of disk arms that are required to sustain the workload in the base case (without replication) and D_{alt} denotes the corresponding number in our alternating mirroring scheme. The average time t_{ran} to serve a random I/O request in the base case involves an average seek time, one half rotation t_{hrot} on the average

to position the heads over the block and the block transfer time t_{xfer}:

$$t_{ran} = t_{avgseek} + t_{hrot} + t_{xfer}$$

The utilization of a disk arm in the base case is:

$$U_{base} = \lambda t_{ran} / D_{base}$$

In our mirroring scheme, reads require the same amount of time as in the base case. Writes, however, take less time, since they can take advantage of the batching effect. Let t_{amw} denote the amortized time for a block write (we estimate t_{amw} below). Then, the utilization in the alternating mirroring case is:

$$U_{alt} = \frac{\lambda(1-w)t_{ran} + 2\lambda w t_{amw}}{D_{alt}}$$

The factor of two comes from the fact that writes must be performed twice (once for each disk).

For the comparison to be fair, we require that the disks be operated at the same load in both cases, so by equating U_{base} and U_{alt} we obtain the *efficiency ratio* of the two schemes, i.e., the ratio of the disk arms required in each case:

$$\frac{D_{base}}{D_{alt}} = \frac{1}{1 + w(2\frac{t_{amw}}{t_{ran}} - 1)}$$

Intuitively, the lower the efficiency ratio, the more the arms required by the alternating scheme to replace a single base case arm. Note that if $t_{amw}/t_{ran} < 1/2$ (which we show is actually the case), then the efficiency ratio is higher than 1.

We now estimate t_{amw}. Assuming that a fraction f of the disk space can be accommodated in main memory, the batch size is:

$$batch\_size = f * blks_{trk} * surfs * cyls$$

Assuming a uniform access pattern, the average number of cylinders for which there will be at least one updated block in the batch is:

$$cylinders\_hit = cyls * (1 - (1 - 1/cyls)^{batch\_size})$$

In Reference [17], it is shown that with seek times improving, rotational delays should be taken into account when scheduling a batch of writes. However, for simplicity, we assume that a SCAN policy is used for installing updates, i.e., all updates are applied to a particular cylinder before the heads move to the next cylinder. This is a conservative assumption, since a more efficient batch installation scheme would benefit our mirroring scheme even further. The average number of block updates for each of the *cylinders_hit* is $blocks\_amortized = batch\_size/cylinders\_hit$. The seek time for the Fujitsu disk can be approximated [17] by the function $4.6 + .87\sqrt{seek\_distance}$. Since the seek time is a concave function of the seek distance, we get a conservative (pessimistic) estimate if we approximate the average seek time with the seek time for the average distance. Thus, in the average case, the seek time for moving to the next cylinder where updates must be installed is:

$$t_{seeknext} \simeq 4.6 + .87\sqrt{cyls/cylinders\_hit}$$

The time to write the blocks in a particular cylinder is the time to seek to that cylinder plus the time to rotate enough times to apply all modified blocks on the cylinder. For example, if only one block needs to be written, half a rotation is necessary on the average to get to the beginning of the block, plus the time necessary to transfer the data. If multiple blocks must be written in one cylinder, then there is the possibility of collisions, i.e., two or more blocks may have to be written on different tracks, but on sectors that are vertically aligned, so that they cannot be written during the same rotation. Let us call all vertically aligned blocks a *collision group*. If one assumes zero head switch time, during each rotation one block from each collision group can be written, so the number of rotations required is determined by the maximum cardinality of any collision group. However, zero head switch times are unrealistic, so we impose the *constraint P*, that if a block is written on one track, the heads are unable to write a block on a *different* track at the next vertical position during the same rotation. In Section 3.2 we compare the results without constraint P (optimistic case) with the results under constraint P (pessimistic case).

Given the above constraint, it is hard to determine the optimal policy for installing updates in a cylinder. Thus, we use a greedy heuristic that keeps installing *consecutive* blocks on the same track as long as it can; when the run of consecutive blocks is exhausted, it switches to a different track. The *number_of_rotations* required by this heuristic is difficult to estimate analytically, so we use a Monte Carlo simulator as a subroutine in our analytical computations. The simulator takes as input the number of blocks to be written in a cylinder.

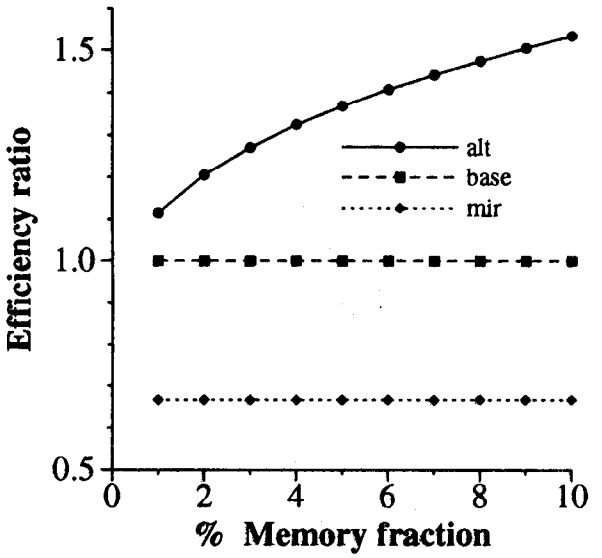

Figure 3: Efficiency vs. memory fraction

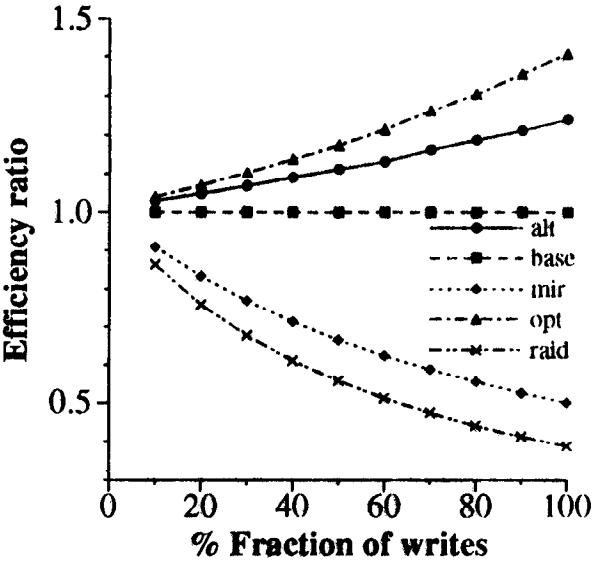

Figure 4: Efficiency vs. fraction of write requests

In each step, the simulator generates a random assignment of blocks to locations on the cylinder and calculates the number of rotations that would be required for that assignment. This step is repeated and the result is averaged over all iterations.

Then, we can calculate t_{amw}

$$t_{amw} = \frac{t_{seeknext} + number\_of\_rotations * t_{rot}}{blocks\_amortized}$$

and from that, the efficiency ratio. In order to be able to compare our analytical results with the simulation results reported in the following section, the efficiency ratio must be multiplied by the corrective factor $\frac{1}{1-f}$ to account for the fact that since memory contains a fraction f of the data, an equal fraction of access requests will get a hit in memory and will not cause disk I/O. (Again, this is conservative, since it assumes uniform access.) Thus, the formula for the efficiency ratio becomes:

$$\frac{D_{base}}{D_{alt}} = \frac{1}{1 + w(2\frac{t_{amw}}{t_{ran}} - 1)} \times \frac{1}{1-f}$$

In Figure 3 we show the efficiency ratio for the alternating mirroring scheme (solid), the base non-redundant scheme (dashed) and the plain mirroring scheme (dotted) as the memory expressed as a fraction f of the disk space varies from 1% to 10% (the write fraction is kept at the default value of 0.5). In plain mirroring, reads are serviced by one disk, while modified blocks cause a random write at each disk of a mirrored pair. We observe that alternating mirroring is more efficient and requires *fewer* arms than the base scheme in the entire range presented. This is because the increased efficiency gained from *all* writes more than compensates for the extra writes introduced. As the memory fraction increases, batches become larger, amortized seek times are reduced, but the probability of collisions increases, so that the expected number of rotations increases. As shown in Figure 3, the seek time reduction dominates, writes become more efficient for larger batches and the performance difference becomes increasingly favorable for mirroring. Thus, our technique will benefit from the advent of large main memories.

As memory sizes increase, buffer pools become larger and can capture a significant fraction of the read traffic. Thus, in the future we expect to see the I/O traffic directed to disks consisting mainly of writes. Figure 4 plots the efficiency ratio of the alternating mirroring scheme (solid), the base non-redundant scheme (dashed) and the plain mirroring scheme (dotted) against the fraction of write requests in the system. The memory size is held constant at 1% of the disk size. As the write ratio increases, the performance of alternating mirroring *improves*, since an increasing fraction of requests can be performed in batch mode. This is a novel result, which contrasts with the widely held belief that mirroring is arm costly for loads with intensive write traffic. This belief is true only for the

plain mirroring scheme, as shown in Figure 4.

In Figure 4 we show two more curves. The curve labelled "opt" corresponds to the alternating mirroring scheme under the optimistic assumption that head switch is infinitely fast. In practice, this may be achieved by formatting the disk with a slight offset between tracks, which leads to a spiral arrangement. Thus, there may be enough time to switch heads without missing the next vertical position. In this case, the *number_of_rotations* equals the maximum number of block writes hitting a vertical position. As shown in Figure 4, this assumption leads to even higher efficiency ratio for the alternating scheme.

For comparison purposes, the curve labelled "raid" in Figure 4 shows the efficiency ratio for the RAID-5 scheme. The efficiency ratio of RAID-5 is below that of the other schemes in the entire range. The RAID-5 curve is for a best-case scenario, where the old copy of a modified data block is available in memory, so that a logical write involves a write to the data block and a read-modify-write to the checksum. In practice, the old value of the data block may also have to be read, causing a read-modify-write for both the data and the checksum and resulting in even lower efficiency for RAID-5.

Finally, we study the effect of disk parameters. Although seek times and rotational delays have not improved significantly, disk densities (and capacities) increase continuously, so it is interesting to investigate the impact of this trend. Figure 5 plots the efficiency ratio of the three schemes as the number of blocks per track varies from 8 to 16. The write fraction is held constant at 0.5, while memory is 1% of the corresponding disk size at each data point. The increase in density affects the ratio for alternating mirroring favorably, because the performance difference between sequential and random I/O increases.

If one assumes that memory is free, i.e., our scheme can take advantage of an already existing buffer pool, then the efficiency ratio becomes also a measure of cost-efficiency. Under the assumption of free memory, the efficiency of the other schemes would improve due to buffer cache hits, but it would still be below that of alternating mirrors, so that alternating mirrors offer reliability as well as the best price-performance, even compared with non-redundant schemes.

Let us now examine the case when memory is introduced for exclusive use by the alternating mirrors, so that the alternating mirrors get charged

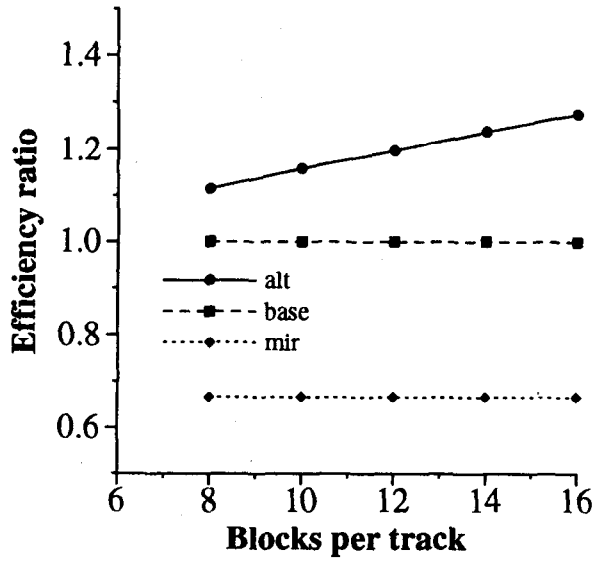

Figure 5: Efficiency vs. number of blocks per track

Parameter	Value
Price/disk	$3000
Memory Price/MB	$250
Disk Size	1200MB
Disk Controller Price	$1000
Random I/Os / sec / disk	35

Table 2: Price Parameters

with its cost. Using the price parameters shown in Table 2, Figure 6 plots the cost per I/O operation in the alternating scheme as the memory fraction of disk space increases. In this range of parameters, the cost increases monotonically with memory, since the cost of memory dominates the disk cost. As mentioned above, increased memory size implies increased performance for the alternating mirroring scheme, which tends to decrease the cost per I/O operation. On the other hand, the addition of memory increases the cost of the system, so we expect to see a minimum in the price per I/O per sec in our scheme. Figure 7 plots the cost per I/O per sec varying the memory fraction f in the range of $f = 0.01\%$ to $f = 1\%$. The minimum appears at $f = 0.06\%$ and is $107 per I/O per sec. For comparison purposes, the cost of the base scheme is $94 per I/O per sec and that of plain mirroring $141 per I/O per sec. The reason the minimum appears

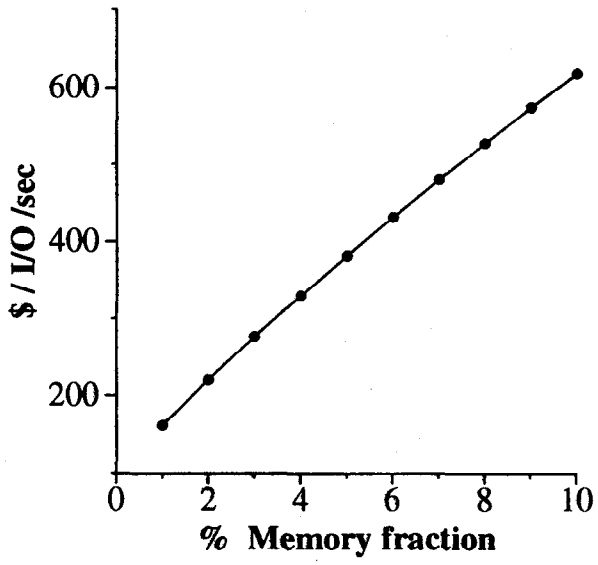

Figure 6: Cost per I/O vs. memory fraction

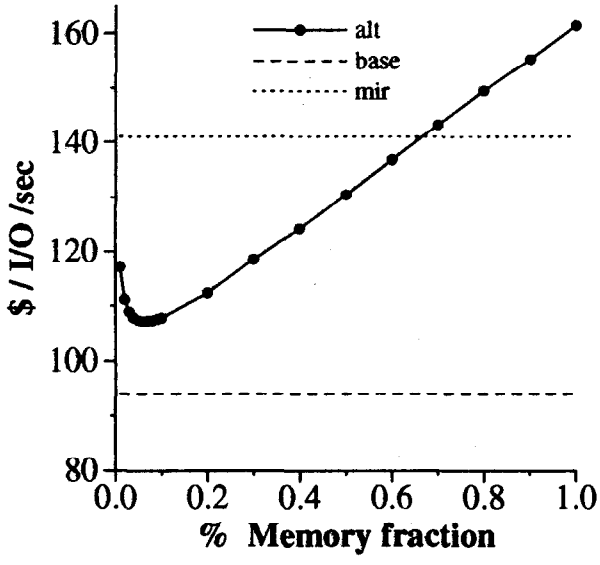

Figure 7: Cost per I/O vs. memory fraction

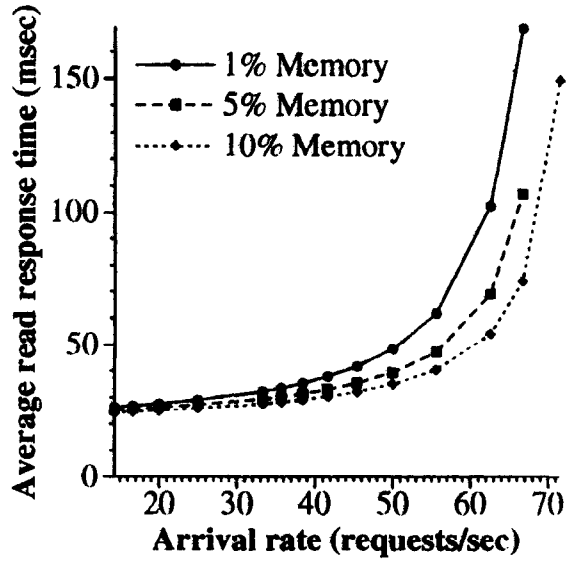

Figure 8: Read response time vs. throughput for different memory sizes

at such low memory size is that we have assumed a relatively high price for memory ($250/MB). As the cost of memory comes down, the minimum will move to the right and the cost of the alternating scheme will drop significantly in the entire range. It is for this reason that in the analysis we focused our attention on larger memories.

Note that the performance of alternating mirroring may actually be better than what the analysis above indicates. In our analysis we assumed that disk utilization is the same in all schemes. However, alternating mirroring has a better response time for reads than the base scheme, which makes it possible to operate alternating mirrors at higher utilization and still perceive the same performance as the base case. Higher utilization implies that the number of arms can be further reduced, yielding higher efficiency ratio, lower cost per I/O per sec, etc.

3.2 Simulation Results

To verify the analytical results of the previous section, we built an event driven simulator for the proposed mirroring scheme with deferred updates. In Figure 8 we plot the response time for read operations as the arrival rate increases. The workload is 50% reads and 50% writes. The three curves correspond to three different sizes of main memory: 1% of disk space (solid line), 5% of disk space

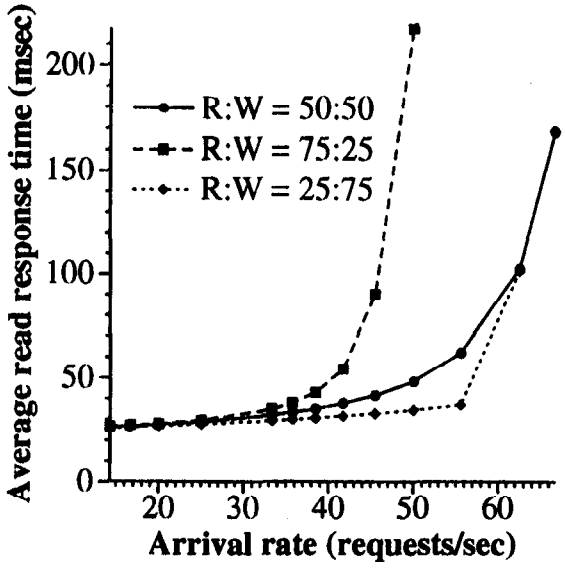

Figure 9: Read response time vs. throughput for different write ratios

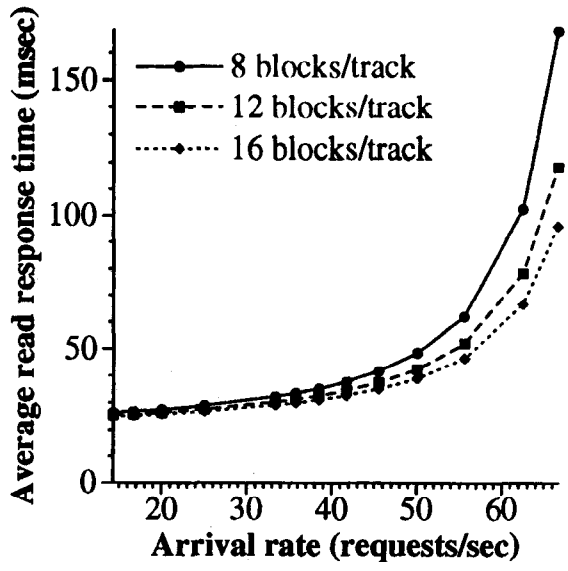

Figure 10: Read response time vs. throughput for different disk densities

(dashed line) and 10% of disk space (dotted line). As memory increases, so does the maximum load that can be sustained by the system. Dividing the maximum throughput measured for a pair of disks employing the proposed scheme by twice the maximum throughput of a disk serving random requests we get a simulation estimate for the efficiency ratio of the proposed scheme. (Alternatively, the disk arm utilizations can be compared.) The simulation values for the efficiency ratio are 1.11 (for 1% memory), 1.35 (for 5% memory) and 1.5 (for 10% memory). Comparing these values with the values in Figure 3 we observe a very good match between analysis and simulation across the entire range.

Note that, for stability reasons, real systems are typically configured to operate at an average throughput well below the saturation point (sometimes as low as 3 standard deviations below the average arrival rate that would yield acceptable response time [11]). However, the arrival rate that saturates the system is a good means of comparison, since the operating point is derived from it. As the saturation point increases, so does the point of normal operation.

In Figure 9 we plot the read response time against throughput for different mixes of read and write operations. The three curves correspond to 75% reads – 25% writes (dashed line), 50% reads – 50% writes (solid line) and 25% reads – 75% writes (dotted line). As the write ratio increases, so does the maximum throughput sustainable by the proposed scheme. This is because two batched writes are cheaper than a random read. The efficiency ratios measured by the simulation were 1.05 (25% writes), 1.11 (50% writes) and 1.18 (75% writes), which match very well with the values computed analytically and shown in Figure 4.

In Figure 10 we plot the read response time against throughput for different numbers of blocks per track on the disk. Memory is 1% of disk size and write fraction is 0.5. The three curves correspond to 8 blocks/track (solid line), 12 blocks/track (dashed line) and 16 blocks/track (dotted line). Again, as the disk density increases, so does the maximum throughput sustainable by the proposed scheme, since the efficiency of batched writes increases. The efficiency ratios measured by the simulation were 1.11 (8 blocks/track), 1.20 (12 blocks/track) and 1.27 (16 blocks/track), which match very well with the values computed analytically and shown in Figure 5.

4 Recovery

We consider recovery from disk failures. We assume that other kinds of failures (e.g., processor failures) are handled with conventional recovery techniques (e.g., replay of redo logs, processor takeover), and are not the thrust of this paper.

Suppose that a disk fails. A recovery procedure must ensure continuous access to the data that was stored on the failed disk, so that transaction processing will not be interrupted. Furthermore, the lost data must be restored on a replacement disk. If we define as failure of the disk system the case when some data is lost and cannot be retrieved even by the redundancy mechanism, it has been shown [5] that the mean time to failure of a disk system is inversely proportional to the restoration time of a single failed disk. Thus, the failed disk must be restored as quickly as possible.

In RAID systems, recovery is a complicated process: in order to reconstruct a block that was on the failed disk, one block must be read from each of the remaining disks of the group. In the worst case, all requests are reads and each of the surviving disks must serve twice as many requests as during normal processing, which hurts response time and limits throughput. Even if the transaction load is not entirely reads, the background process that reconstructs cold blocks still imposes a significant overhead on the system. For the restoration process to finish within a reasonable time and avoid hurting the system's MTTF, degraded performance must be tolerated, as is indicated by studies that have dealt with RAID performance under failure [12, 13, 14].

The performance of mirroring under failure has been studied in Reference [5], which emphasizes the effect of data placement on recovery time. The surviving disk serves interactive traffic and also helps restore the data on the replacement disk. It is assumed that the system will operate in degraded mode during recovery.

There are already installations with over a thousand disks, and it is predicted [10] that if current trends in disk technology continue, we will soon see installations with thousands of disks. With such numbers of disks, failures will be so frequent, that performance degradation during failures may no longer be tolerable. We can extend our ideas for normal processing to the failure scenario and obtain an efficient recovery mechanism that can guarantee good performance for the entire duration of recovery.

During normal operation in our scheme there is always at least one disk arm dedicated to serving read requests in any mirrored pair. There may be periods when both arms serve reads, but the guaranteed capacity dedicated to reads is one arm, since the other arm may be installing a write batch. Our recovery technique guarantees that at least one

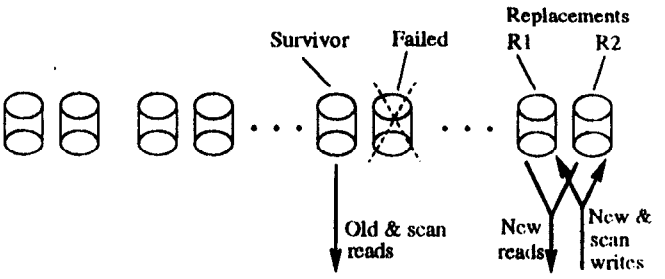

Figure 11: Configuration during recovery.

disk arm's capacity is constantly available to serve random read requests to the data stored on the *surviving disk throughout the recovery period.* Previous recovery techniques cannot guarantee that one disk arm's capacity is dedicated to random reads during recovery. In our scheme the same level of service guarantee holds during normal operation and recovery.

As is typical for installations with many disks, we assume that there are a few spare disks available, so that recovery can start immediately upon detection of failure. Since recovery is arm bound, our recovery procedure uses *two* spare disks, designated as the *replacement* disks R_1 and R_2. At the end of the recovery phase, each of the replacement disks holds a complete, up-to-date copy of the data. The two replacement disks form the new mirrored pair, while the survivor is returned to the system as a spare disk.

The recovery process is not instantaneous, so blocks will be updated during the recovery phase. Since the survivor disk has a high read load (random reads plus restoration reads), we relieve it of any write effort by redirecting all updates to the replacement disks only. The survivor disk is kept in read-only mode until the end of recovery, which implies that the survivor will become out-of-date, but this is not a problem, since the disk will be returned to the system, anyway.

To achieve write efficiency, the replacement disks alternate between read-only and write-only modes to install updates and cold data scanned by the survivor. The configuration during recovery is shown in Figure 11.

The replacement disks both install all modified blocks. This ensures that there is always a complete, up-to-date copy of the data to service random read requests: when replacement disk R_1 is in write-only mode, the up-to-date copy consists of the survivor and replacement R_2, with blocks on replacement R_2 superseding older versions on the

survivor. When replacement R_2 is in write-only mode, the up-to-date copy consists of the survivor and replacement R_1, with blocks on replacement R_1 superseding older versions on the survivor.

The two replacement disks are initially blank. They write the blocks that are available in the memory buffer. For the duration of recovery, they alternate between periods of read-only and write-only activity, but are kept out of phase. During periods of read-only activity, they provide access to data they have written. When a read request arrives in the system, if the data is available on the replacement disk which is in read-only mode at that moment, the request is routed to that disk, otherwise to the survivor disk. A block directory is used to determine if a block is available on a replacement disk. The directory gets updated as blocks are written on the replacement disks. As the replacement disks take on an increasing part of the read traffic, the survivor disk has spare capacity to perform a scan of the cold data.

The survivor can also use an opportunistic strategy to scan data. For example, blocks that pass under the head during the rotational delay of a random read can be read at zero cost. Many disk drives already use track buffers to provide this capability.

Data accumulates in the main memory buffer as a result of random reads to unscanned data, updates and scanning. Periodically, when there is enough data for a batch, one replacement disk enters a write-only mode to write the available data. During this time, the other replacement provides access to data that has been modified since the beginning of the recovery (and has been missed by the survivor) as well as to unmodified data that has already been scanned. When one replacement disk finishes writing the batch, the roles of the replacements are reversed and the other replacement writes the batch.

We use a simple model to evaluate recovery performance under this scheme. We assume that when the failure occurs, transaction processing is not suspended and the survivor disk receives a continuous stream of random read requests at the maximum rate sustainable during normal processing, i.e., $1/t_{ran}$. This random access rate, expressed as a disk fraction, is

$$r_r = 1/(t_{ran} * cyls * blks_{trk} * surfs)$$

As the replacement disks start taking on some read activity, some of the random read slots become available for scanning data. This involves (possibly) seeking to a nearby cylinder that has unscanned data and then reading as many blocks as possible in the slot, say g blocks.

In the beginning of the process, all cylinders have unscanned data, so the scan does not require a seek. If the access pattern is uniform, all cylinders will be scanned at the same rate and no seeks will be necessary approximately until the time that cylinders are left with one unscanned track. For our parameters (20 tracks per cylinder), this means that no seeks are necessary for 95% of the scan. As we go beyond that point, the seeks get longer, but their number gets smaller. We amortize the cost of the long seeks by estimating the expected seek distance $seek_{scan}(u)$ when a fraction u of the data is unscanned and then integrating $seek_{scan}(u)$ (continuous approximation) in the interval $[0, 1]$.

Let u be the fraction of unscanned data at some point in time. Then, the average number of cylinders that contain unscanned tracks is approximately

$$non\_empty\_cyl \simeq cyls*(1-(1-1/cyls)^{u*total\_tracks})$$

where $total\_tracks$ is the total number of tracks in the disk

$$total\_tracks = cyls * surfs$$

The average distance between two successive cylinders with unscanned data is approximately

$$\begin{aligned} k &\simeq \frac{cyls}{non\_empty\_cyl} \\ &\simeq (1-(1-1/cyls)^{u*total\_tracks})^{-1} \end{aligned}$$

We assume that the head always lies between two cylinders with unscanned data and seeks to the closest of the two cylinders. We ignore the end effect, where the head can only seek in one direction, because there is no unscanned data in the other direction. If we assume that the head is positioned with equal probability at any position between the two cylinders with unscanned data, the expected seek distance to the closest cylinder is approximately $seek_{scan} \simeq (1+2+\cdots+k/2)/(k/2) \simeq 0.25k$. Thus,

$$seek_{scan}(u) \simeq 0.25(1-(1-1/cyls)^{u*total\_tracks})^{-1}$$

We estimate the amortized seek distance for each scan step with the definite integral

$$\int_0^1 seek_{scan}(u)\,du = u - \frac{\ln(1-(1-\frac{1}{cyls})^{u*total\_tracks})}{total\_tracks \ln(1-\frac{1}{cyls})}\Bigg|_0^1$$

Since the integral of $seek_{scan}$ is not defined for $u = 0$, we actually use $1/total\_tracks$ as the lower limit. For our parameters, the result of the integration is 0.33 cylinders. Given this result and the fact that the random read takes 28.4 msec on the average, we get a conservative estimate if we assume that the head always seeks a distance of one before scanning. This seek takes 5.47 msec, which leaves 23 msec for rotation. In a single rotation (16.66 msec) 8 blocks can be read, and head switching takes roughly 1 msec, so assuming that the number of blocks read in a scan slot (corresponding to the random read) is $g = 8$ to 10 seems a conservative choice (g is higher at the beginning of the scan and lower towards the end). The access rate of scanning expressed as a disk fraction is $r_s = gr_r$.

Assume that at some point in time a fraction x of the data is available on the replacement disk. If we assume a uniform access pattern, a fraction x of the read requests is directed to the replacement disk. This means that on the survivor disk a fraction x of the slots is available to the scanning process, while the remaining $(1 - x)$ fraction of the slots are random read slots. The random read slots are not useless to the restoration process, since they are read requests for uncopied data (otherwise the request would have been routed to the replacement disk operating in read-only mode). Thus, during a fraction x of the time uncopied data is encountered at rate r_s and during a fraction $1-x$ of the time at rate r_r, so the effective rate is $xr_s+(1-x)r_r$. Since there are a large number of blocks, we approximate the problem with its continuous case and solve the equation

$$\frac{dx}{dt} = xr_s + (1 - x)r_r$$

with initial condition $x(0) = f$, since the data in the buffer pool is available immediately to the replacement disk. The above equation has the solution:

$$x(t) = (f + \frac{1}{g-1})e^{(g-1)r_r t} - \frac{1}{g-1}$$

To find the time it takes to scan the entire survivor disk we solve for $x = 1$ and get:

$$t = \frac{\ln(g) - \ln(fg - f + 1)}{(g-1)r_r}$$

Figure 12 plots the recovery time in minutes against g, for $f = 0.01$ (solid), $f = 0.05$ (dashed) and $f = 0.10$ (dotted). For $g = 8$ and $f = 0.01$, $t = 18$ min, while for $g = 10$ and $f = 0.05$, $t = 13$ min.

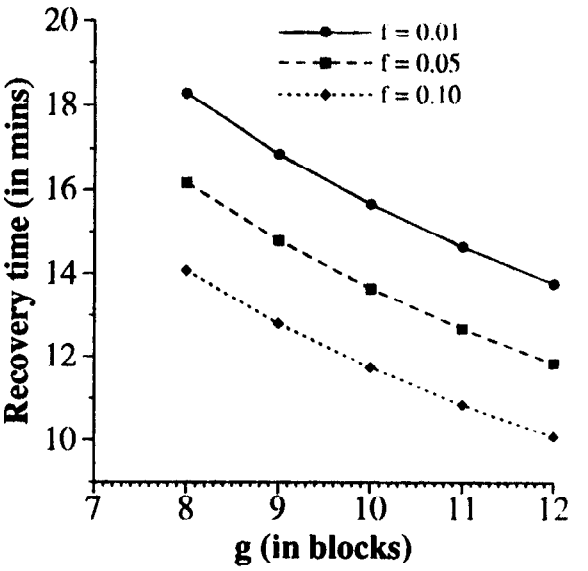

Figure 12: Recovery time vs. scan rate for different memory sizes

If $mttf_s$ is the mean time to failure for a single disk drive and t_{rec} is the recovery time, the mean time to failure $mttf_{sys}$ (i.e., loss of both copies of some data) for a system with N disks ($N/2$ pairs) is [5]:

$$mttf_{sys} = \frac{mttf_s^2}{Nt_{rec}}$$

If we take $mttf_s = 30,000$ hrs and $t_{rec} = .25$ hrs, for a system with 1,000 disks $mttf_{sys} = 410$ yrs.

Our estimates for recovery time are fairly conservative. The number of blocks that can be read in a scan slot is underestimated. The free reads that can be obtained while waiting for the rotation of the random reads have not been taken into account. A uniform access pattern has been assumed. If the actual access pattern is skewed, a small amount of hot data receives a disproportionately high amount of traffic. The hot data will be "filtered" out of the survivor disk quickly and the replacement disks will take on a substantial fraction of the read traffic quickly, so that more scanning capacity will become available on the survivor sooner and recovery time will be shorter.

In a variation of the proposed recovery scheme, replacement disk R_1 can write scan blocks as well as modified blocks, while replacement disk R_2 writes only the modified blocks (not the scanned

blocks). The two replacement disks alternate between write-only and read-only periods and the survivor is in read-only mode. When a complete, up-to-date version of the data is created on R_1, the modified blocks that have accumulated on R_2 are copied to the survivor, so that the survivor and R_1 become the new mirrored pair, while R_2 is returned to the system. This option would allow R_2 to be of smaller capacity, since it would have to store only modified blocks. However, while R_1 is in write-only mode, only reads to modified blocks are rerouted to R_2; reads to already scanned data must be serviced by the survivor, which would lower the scan rate at the survivor. In case of a skewed access pattern, most of the accesses may go to modified data, so the scan degradation may not be significant.

5 Discussion

We have presented a simple scheme for scheduling I/O requests in a mirrored disks environment, which achieves high disk bandwidth during normal operation. We have also presented an efficient scheme for recovering from single disk failures. The main observation is that, in contrast to current belief, mirroring is not expensive, since it can operate at better arm efficiency than even non-redundant schemes. The bandwidth saved can be used to support more data under the disk arm. For example, higher space utilization can be obtained in disk arm bound systems. Alternatively, higher I/O bandwidth can be provided to hot data in applications for which the bandwidth of non-redundant schemes is inadequate.

In previous sections we only considered the case of non-overlapping write-only periods for the two disks of a mirrored pair. We now determine the parameter range in which no overlapping occurs. The first condition is that there should be enough time to service the read operations. If each disk spends half its time in write-only mode, the total disk time available to service random reads during a period T is T. The number of read requests received during a period T is $\lambda(1-w)T$ and the time required to service them is $\lambda(1-w)Tt_{ran}$. Thus, $\lambda(1-w)Tt_{ran} \leq T$, or $w \geq 1-1/(\lambda t_{ran})$. The second condition is that each disk should be able to write a batch of size λwT (which is the number of writes accumulating during a period) in time less than $T/2$, or $\lambda wTt_{amw} \leq T/2$, or $w \leq 1/(2\lambda t_{amw})$. Recall from Section 3.1 that the value of t_{amw} depends on the memory size. In Figure 13 we plot the two curves corresponding to the above constraints for w as a function of the arrival rate λ, with memory held constant at 1% of disk size. The response time curves of Section 3.2 show that the throughput must generally be kept below 40 requests per second to obtain good response time. In Figure 13 we observe that essentially no overlapping occurs for such arrival rates, which justifies our focus on non-overlapping write-only periods in the previous sections. Furthermore, as the size of the memory increases, the batching effect becomes stronger, writes become more efficient, and the feasible region expands.

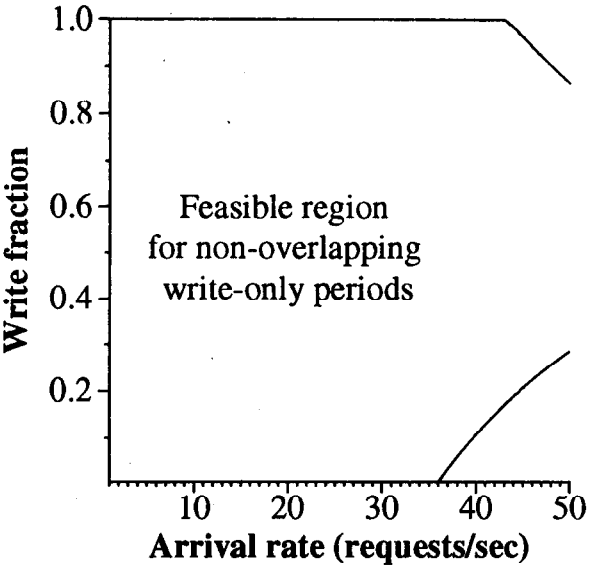

Figure 13: Write fraction range for non-overlap vs. arrival rate

For sequential write intensive operations (e.g., load, copy), both disks will spend all of their time applying writes in batches. Throughput will be as good as in the case of plain mirrors applying writes in synchrony. However, since in alternating mirroring one of the disks lags half a cycle behind the other, the memory necessary to buffer the writes is wasted (plain mirroring would not need this memory).

The batching of write requests used in our alternating scheduling policy cannot be directly applied to non-redundant disk systems because it would hurt the response time for random reads. In Figure 14 we plot the read response time against the request arrival rate for a single disk. The workload is 50% reads – 50% writes. Write requests accumulate in memory (1% of disk size). When the

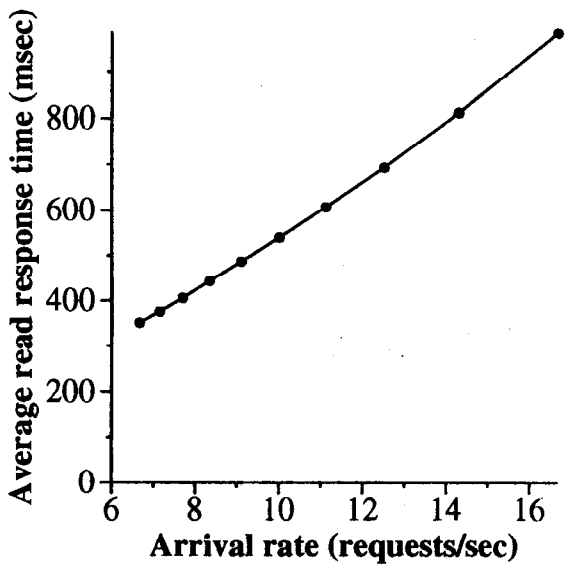

Figure 14: Read response time vs. arrival rate for single disk batching

memory fills up, the writes are applied in batch mode. As shown in Figure 14, the average response time deteriorates significantly, even for low arrival rates. The variance of the read response time is enormous, since some requests are serviced immediately, while other requests must wait for an ongoing batch write to complete. For example, for the lowest arrival rate shown in Figure 14 (6.66 requests per second), the average response time for reads is 352 msec, while the standard deviation is 1780 msec. By comparison, Figure 8 shows that the read response time for alternate mirroring stays below 30 msec, even for arrival rates as high as 25 requests per second.

The alternating technique is not applicable to other redundancy schemes (e.g., RAID) either, because the redundant data is not usable by applications. However, Reference [1] shows that the batching effect can be exploited to increase the efficiency of writing checksum data.

In our discussion, we focused on transaction processing systems that employ a write-ahead log. For other random access bound applications, nonvolatile, reliable memory can be used. For example, the buffer pool can be stored on the safe RAM described in Reference [6].

Another issue related to memory is that our scheme can use a general purpose buffer pool to accumulate batches. This is in contrast to schemes that may need memory for data not otherwise usable [1].

Acknowledgement

We thank Jim Gray and the anonymous referees for many insightful suggestions that improved the paper significantly.

References

[1] A. K. Bhide, D. M. Dias, and C. A. Polyzois. Reliable disk array architectures for transaction processing. Submitted for publication.

[2] D. Bitton. Arm scheduling in shadowed disks. In *IEEE Compcon*, pages 132–136, San Francisco, California, February 1989.

[3] D. Bitton and J. Gray. Disk shadowing. In *14th Int'l Conf. on Very Large Data Bases*, pages 331–338, Los Angeles, California, August 1988.

[4] P. M. Chen, G. A. Gibson, R. H. Katz, and D. A. Patterson. An evaluation of redundant arrays of disks using an Amdahl 5890. In *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 74–85, Boulder, Colorado, May 1990.

[5] G. Copeland and T. Keller. A comparison of high-availability media recovery techniques. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 98–109, Portland, Oregon, June 1989.

[6] G. Copeland, T. Keller, R. Krishnamurthy, and M. Smith. The case for safe RAM. In *15th Int'l Conf. on Very Large Data Bases*, pages 327–335, Amsterdam, August 1989.

[7] Fujitsu Limited. *M2361A Minidisk Drive Engineering Specifications*, 1984.

[8] H. Garcia-Molina and C. A. Polyzois. Processing of read-only queries at a remote backup. Technical Report CS-TR-354-91, Department of Computer Science, Princeton University, December 1991.

[9] J. P. Gelb. System-managed storage. *IBM Systems Journal*, 28(1):77–103, 1989.

[10] J. Gray, B. Horst, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *16th Int'l Conf. on Very Large Data Bases*, pages 148–161, Brisbane, Australia, August 1990.

[11] B. McNutt. DASD configuration planning: Three simple checks. In *Int'l Conference for the Management and Performance Evaluation of Computer Systems*, pages 990–997, Orlando, Florida, December 1990.

[12] J. Menon and D. Mattson. Comparison of sparing alternatives for disk arrays. In *19th Int'l Symposium on Computer Architecture*, pages 318–329, Gold Coast, Australia, May 1992.

[13] J. Menon and D. Mattson. Distributed sparing in disk arrays. In *IEEE Compcon*, pages 410–421, San Francisco, California, February 1992.

[14] R. R. Muntz and J. C. S. Lui. Performance analysis of disk arrays under failure. In *16th Int'l Conf. on Very Large Data Bases*, pages 162–173, Brisbane, Australia, August 1990.

[15] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 109–116, Chicago, IL, June 1988.

[16] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, February 1992.

[17] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *Winter 1990 USENIX*, pages 313–323, Washington, D.C., January 1990.

[18] J. A. Solworth and C. U. Orji. Write-only disk caches. In *ACM SIGMOD Int'l Conf. on Management of Data*, pages 123–132, Atlantic City, New Jersey, June 1990.

[19] J. A. Solworth and C. U. Orji. Distorted mirrors. In *First Int'l Conf. on Parallel and Distributed Information Systems*, pages 10–17, Miami Beach, Florida, December 1991.